



PATENT ABSTRACTS OF JAPAN

(11) Publication number: **11212456 A**

(43) Date of publication of application: 06.08.99

(51) Int. Cl. **G09C 1/00**
G06F 7/72
H04L 9/30

(21) Application number: **10014681**(22) Date of filing: **27.01.98**(71) Applicant: **FUJITSU LTD**

(72) Inventor: **TAKENAKA MASAHIKO**
ITO KOICHI
TORII NAOYA

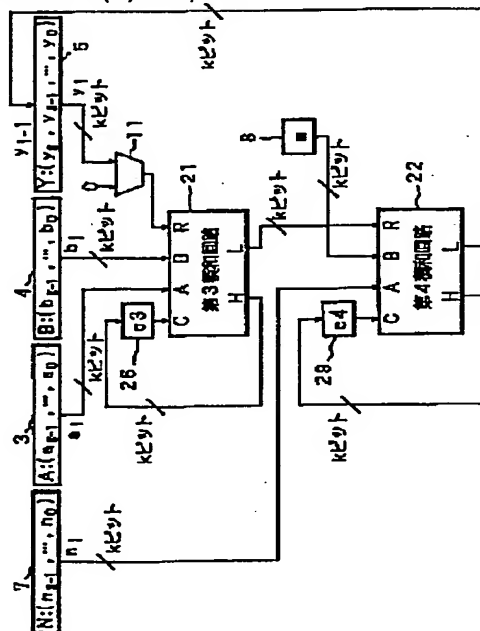
(54) **MULTIPLICATION REMAINDER CALCULATION
 DEVICE USING MONTGOMERY METHOD**

COPYRIGHT: (C)1999,JPO

(57) Abstract:

PROBLEM TO BE SOLVED: To provide a calculation device which simplifies the constitution of a product sum circuit and permits pipeline processing and uses a Montgomery algorithm to quickly perform multiplication remainder calculation.

SOLUTION: A product sum circuit 21 multiplies outputs of an A register 3 and a B register 4 and adds outputs of a c3 register 26 and a Y register 5 to the multiplication result. A product sum circuit 22 multiplies outputs of an N register 7 and an m register 8 and adds outputs of a c4 register 29 and the product sum circuit 21 to the multiplication result. Registers 26 and 29 for carry in two product sum circuits 21 and 22 are provided independently of each other, and carry is returned to the corresponding product sum circuit. All the processing is performed in a processing unit (k bits). The next operation of the product sum circuit 21 can be performed during the operation of the product sum circuit 22.



(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平11-212456

(43) 公開日 平成11年(1999) 8月6日

(51) Int.Cl.⁸
 G 0 9 C 1/00
 G 0 6 F 7/72
 H 0 4 L 9/30

識別記号
 6 5 0

F I
 G 0 9 C 1/00 6 5 0 A
 G 0 6 F 7/72
 H 0 4 L 9/00 6 6 3 B

審査請求 未請求 請求項の数13 O L (全 16 頁)

(21) 出願番号 特願平10-14681

(22) 出願日 平成10年(1998) 1月27日

(71) 出願人 000005223

富士通株式会社

神奈川県川崎市中原区上小田中4丁目1番
1号

(72) 発明者 武仲 正彦

神奈川県川崎市中原区上小田中4丁目1番
1号 富士通株式会社内

(72) 発明者 伊藤 孝一

神奈川県川崎市中原区上小田中4丁目1番
1号 富士通株式会社内

(72) 発明者 鳥居 直哉

神奈川県川崎市中原区上小田中4丁目1番
1号 富士通株式会社内

(74) 代理人 弁理士 河野 登夫

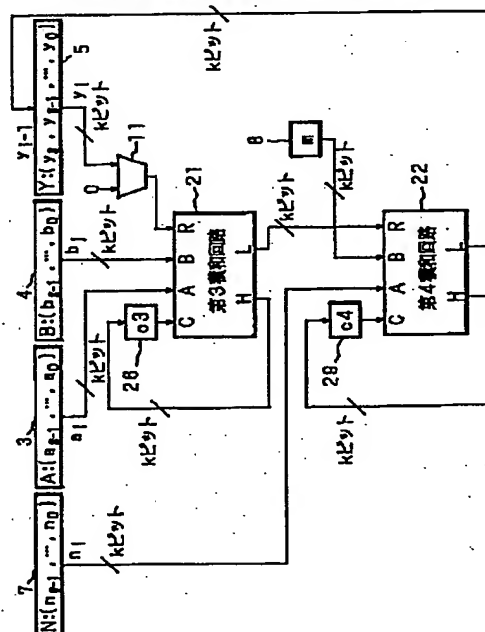
(54) 【発明の名称】 モンゴメリ法による乗算剰余計算装置

(57) 【要約】

【課題】 積和回路の構成を単純化することができ、また、パイプライン処理が可能になる、モンゴメリのアルゴリズムを用いて乗算剰余計算を高速に行う計算装置を提供する。

【解決手段】 積和回路21は、Aレジスタ3及びBレジスタ4の出力を乗算し、その乗算結果にc3レジスタ26の出力及びYレジスタ5の出力を加算する。積和回路22は、Nレジスタ7及びmレジスタ8の出力を乗算し、その乗算結果にc4レジスタ29の出力及び積和回路21の出力を加算する。2つの積和回路21、22におけるキャリー用のレジスタ26、29を各別に設けて自身の積和回路にキャリーを戻す構成とする。全ての処理を処理単位(kビット)内で行う。積和回路22の動作中に、積和回路21の次の動作が可能である。

本発明の乗算剰余計算装置(第2発明)の構成図



【特許請求の範囲】

【請求項 1】 モンゴメリのアルゴリズムを用いて乗算
剰余計算を行う装置において、

積和演算を行いその演算結果を上位 k ビットと下位 k ビットとに分けて出力する第 1 積和回路と、積和演算を行
いその演算結果を上位 k ビットと下位 k ビットとに分けて出力する第 2 積和回路と、加算演算を行いその演算結果を上位 1 ビットと下位 k ビットとに分けて出力する加
算回路と、乗算される 2 数を保持する第 1 及び第 2 レジスタと、前記第 2 積和回路の下位 k ビット出力を保持
し、前記第 2 積和回路のその次の回の下位 k ビット出力を格納する第 3 レジスタと、前記加算回路の下位 k ビ
ット出力を保持し、前記加算回路のその次の回の下位 k ビット出力を格納する第 4 レジスタと、剰余の法を保持す
る第 5 レジスタと、モンゴメリのアルゴリズムにおけるパラメータの値を保持する第 6 レジスタと、前記加算回
路の上位 1 ビット出力を保持し、前記加算回路のその次の回の上位 1 ビット出力を格納する第 7 レジスタとを備
え、

前記第 1 積和回路は、前記第 1 及び第 2 レジスタに保持
された 2 数の所定ビットの値を乗算し、その乗算結果に
前記第 3 レジスタに保持された数の所定ビットの値及び
前記第 4 レジスタに保持された値を加算する演算を行
い、

前記第 2 積和回路は、前記第 5 レジスタに保持された数
の所定ビットの値と前記第 6 レジスタに保持された値と
を乗算し、その乗算結果に前記第 1 積和回路の下位 k ビ
ット出力を加算する演算を行い、

前記加算回路は、前記第 1 積和回路の上位 k ビット出力
と前記第 2 積和回路の上位 k ビット出力と前記第 7 レジ
スタに保持された値とを加算する演算を行うように構成
したことを特徴とするモンゴメリ法による乗算剰余計算
装置。

【請求項 2】 前記第 1 積和回路は、前記第 1 及び第 2
レジスタに保持された 2 数の所定ビットの値を乗算し、
その乗算結果に前記第 3 レジスタに保持された数の所定
ビットの値を加算し、その加算結果に前記第 4 レジスタ
に保持された値を加算するように構成した請求項 1 記載
のモンゴメリ法による乗算剰余計算装置。

【請求項 3】 前記第 1 積和回路は、前記第 1 及び第 2
レジスタに保持された 2 数の所定ビットの値を乗算し、
その乗算結果に前記第 4 レジスタに保持された値を加算
し、その加算結果に前記第 3 レジスタに保持された数の
所定ビットの値を加算するように構成した請求項 1 記載
のモンゴメリ法による乗算剰余計算装置。

【請求項 4】 モンゴメリのアルゴリズムを用いて乗算
剰余計算を行う装置において、

積和演算を行いその演算結果を上位 k ビットと下位 k ビ
ットとに分けて出力する第 1 積和回路と、積和演算を行
いその演算結果を上位 k ビットと下位 k ビットとに分け

て出力する第 2 積和回路と、乗算される 2 数を保持する
第 1 及び第 2 レジスタと、前記第 2 積和回路の下位 k ビ
ット出力を保持し、前記第 2 積和回路のその次の回の下
位 k ビット出力を格納する第 3 レジスタと、前記第 1 積
和回路の上位 k ビット出力を保持し、前記第 1 積和回路
のその次の回の上位 k ビット出力を格納する第 4 レジス
タと、剰余の法を保持する第 5 レジスタと、モンゴメリ
のアルゴリズムにおけるパラメータの値を保持する第 6
レジスタと、前記第 2 積和回路の上位 k ビット出力を保
持し、前記第 2 積和回路のその次の回の上位 k ビット出
力を格納する第 7 レジスタとを備え、

前記第 1 積和回路は、前記第 1 及び第 2 レジスタに保持
された 2 数の所定ビットの値を乗算し、その乗算結果に
前記第 3 レジスタに保持された数の所定ビットの値及び
前記第 4 レジスタに保持された値を加算する演算を行
い、

前記第 2 積和回路は、前記第 5 レジスタに保持された数
の所定ビットの値と前記第 6 レジスタに保持された値と
を乗算し、その乗算結果に前記第 1 積和回路の下位 k ビ
ット出力及び前記第 7 レジスタに保持された値を加算す
る演算を行うように構成したことを特徴とするモンゴメ
リ法による乗算剰余計算装置。

【請求項 5】 前記第 1 積和回路は、前記第 1 及び第 2
レジスタに保持された 2 数の所定ビットの値を乗算し、
その乗算結果に前記第 3 レジスタに保持された数の所定
ビットの値を加算し、その加算結果に前記第 4 レジスタ
に保持された値を加算するように構成した請求項 4 記載
のモンゴメリ法による乗算剰余計算装置。

【請求項 6】 前記第 1 積和回路は、前記第 1 及び第 2
レジスタに保持された 2 数の所定ビットの値を乗算し、
その乗算結果に前記第 4 レジスタに保持された値を加算
し、その加算結果に前記第 3 レジスタに保持された数の
所定ビットの値を加算するように構成した請求項 4 記載
のモンゴメリ法による乗算剰余計算装置。

【請求項 7】 前記第 2 積和回路は、前記第 5 レジスタ
に保持された数の所定ビットの値と前記第 6 レジスタに
保持された値とを乗算し、その乗算結果に前記第 1 積和
回路の下位 k ビット出力を加算し、その加算結果に前記
第 7 レジスタに保持された値を加算するように構成した
請求項 4 記載のモンゴメリ法による乗算剰余計算装置。

【請求項 8】 前記第 2 積和回路は、前記第 5 レジスタ
に保持された数の所定ビットの値と前記第 6 レジスタに
保持された値とを乗算し、その乗算結果に前記第 7 レジ
スタに保持された値を加算し、その加算結果に前記第 1
積和回路の下位 k ビット出力を加算するように構成した
請求項 4 記載のモンゴメリ法による乗算剰余計算装置。

【請求項 9】 前記第 2 積和回路による演算中に、前記
第 1 積和回路によりその次の回の演算を行うように構成
した請求項 4～8 の何れかに記載のモンゴメリ法による
乗算剰余計算装置。

【請求項 10】 暗号化／復号化のためのモンゴメリ法による乗算剰余計算を行う装置において、乗算演算を行う第 1 積和手段と、モンゴメリ剰余演算を行う第 2 積和手段とを備え、前記第 1 積和手段の出力を前記第 2 積和手段の入力とする構成を有し、前記第 2 積和手段が演算を行う間、前記第 1 積和手段で次の回の演算を行うように、パイプライン処理すべく構成したことを特徴とするモンゴメリ法による乗算剰余計算装置。

【請求項 11】 前記第 1 積和手段及び第 2 積和手段にあって、各自身の上位出力を各自身の次の回のキャリア入力とするようにした請求項 10 記載のモンゴメリ法による乗算剰余計算装置。

【請求項 12】 前記第 1 積和手段及び第 2 積和手段における演算量が等しい請求項 10 記載のモンゴメリ法による乗算剰余計算装置。

【請求項 13】 前記第 1 積和手段及び第 2 積和手段は、2 つの k ビットの数を乗算する手段と、その乗算結果に 2 つの k ビットの数を加算する手段とを有する請求項 10 記載のモンゴメリ法による乗算剰余計算装置。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】 本発明は、例えば公開鍵暗号系の RSA 暗号処理において、モンゴメリのアルゴリズム (Module Multiplication Without Trial Division, Peter L. Montgomery, Mathematics of Computation, Volume 44, Number 170, April 1985 pp. 519~528 参照) を用いて乗算剰余計算を高速に行う乗算剰余計算装置に関する。

【0002】

【従来の技術】 近年におけるコンピュータネットワークの発達により、データベースを検索する機会、電子メール、電子ニュース等の電子化された情報をネットワークを経由して送受する機会が急速に増加してきている。更に、これらを利用して、オンラインショッピング等のサービスも提供されつつある。しかし、それに伴って、ネットワーク上の電子化されたデータを盗聴する、改竄する、他人になりすましてサービスを無償で受ける等の問題も指摘されている。特に無線を利用したネットワークにおいては、傍受が容易なためにこれらの問題を防止する対策が望まれている。

【0003】 これらの問題に対して暗号技術を応用した暗号化電子メール、利用者認証システムが提案され、種々のネットワークにも導入されつつある。この意味でコンピュータネットワークにおいては暗号化が必須の技術であるといえる。このような暗号技術の中の一つにデジタル署名即ち認証に適した公開鍵暗号方式があるが、暗号化／復号に大量の処理が必要のために高速化が望まれており、様々な高速化アルゴリズムが発表されている。

【0004】 暗号化方式は、大別すると秘密鍵暗号系と

公開鍵暗号系との二つに分類できる。秘密鍵暗号系は、送信者と受信者とが同じ暗号鍵を持つことにより暗号通信を行う方式である。即ち、秘密鍵暗号系では、あるメッセージを秘密の暗号鍵に基づいて暗号化して相手に送り、受け手はこの暗号鍵を用いて暗号文を復号して元のメッセージに戻して情報入手する。

【0005】 公開鍵暗号系は、送信者が公開されている受信者の公開鍵でメッセージを暗号化して送信し、受信者が自分の秘密鍵でその暗号化メッセージを復号することにより通信を行う方式である。即ち、公開鍵暗号系では、公開鍵は暗号化のための鍵、秘密鍵は公開鍵により暗号化された暗号を復号するための鍵であり、公開鍵で暗号化した暗号は秘密鍵でのみ復号することができる。

【0006】 秘密鍵暗号系では、個人が秘密に保管しなければならない鍵が通信相手の数だけ必要であり、必要な総鍵数は n 人のネットワークの場合 $(n-1)/2$ 個である。また、初めて通信する相手に対しては、何らかの方法で秘密鍵の配送が必要であるという欠点がある。この欠点を解消するために、大規模なネットワークでは鍵管理センタを設置し、センタと間の秘密鍵のみを保管し、暗号通信を行う場合はセンタから送信相手との秘密鍵を得る方法が用いられる。この場合、秘密鍵の総数は n 個となる。

【0007】 一方、公開鍵暗号系では、個人が秘密に保管する鍵は自分の秘密鍵のみであり、必要な総秘密鍵数も n 人のネットワークの場合 n 個である。また、初めて通信する相手に対しては、公開鍵の配送を行えば良く、鍵管理センタを設置して、ユーザの公開鍵を n 個公開簿に登録し、センタから送信相手の公開鍵を得る方法が用いられる。この場合、センタは公開鍵の改竄を防ぐだけで、秘密に保管する必要がない。但し、公開鍵方式は秘密鍵方式に比べて鍵のビット数が大きいので保管に要するファイルサイズは大きくなる。

【0008】 また、認証の場合、秘密鍵暗号系では、例えば送信するメッセージを秘密鍵で圧縮変換し、送信文に付加して送り、受信側では同様に圧縮変換して比較する方式がとられている。しかし、送受信が同じ鍵であるため、受信者は認証データを偽造することができる。これに対して、公開鍵暗号系では、秘密鍵で暗号化することができるのは本人だけであるという特徴を利用する。送信者はメッセージを圧縮変換して秘密鍵で暗号化し、送信文に付加して送り、受信者は送信者の公開鍵で付加されたデータを復号し、同様に圧縮変換したものと比較する方式がとられている。この場合、受信者は不正ができない。

【0009】 このように認証系では公開鍵暗号系の技術は必要不可欠であるといえる。しかし、公開鍵暗号系には、暗号化／復号に大量の処理が必要であるという大きな欠点があるため、一般には処理が速い秘密鍵暗号系をメッセージの暗号化に、公開鍵暗号系は認証用という

ように組み合わせて用いられる場合が多い。

【0010】公開鍵暗号系の中で、現在最も有力なものが1977年にリヴェスト (Rivest) , シャミア (Shamir) 及びエイドルマン (Adlman) の三人によって発明されたRSA暗号である。このRSA暗号の基本原理は次のようなものである。

【0011】(RSAの基本アルゴリズム) 暗号鍵

(e, N) と対応する復号鍵 (d, N) とにおいて、 e と N とは公開鍵であり、 d は秘密鍵である。平文を M 、暗号文を C とすると、暗号化 E と復号 D とのアルゴリズムは次のようにあらわされる。

$$C = E(M) = M^e \bmod N$$

$$M = D(C) = C^d \bmod N$$

但し、 $d \cdot e = 1 \bmod \text{LCM} \{ (p-1), (q-1) \}$

$$N = p \cdot q$$

LCM: 最小公倍数 (lowest common multiple)

p, q は大きな素数

【0012】通常、 e, d, M, N などは1024ビット程度の大きな整数が用いられているので、高速指数計算法を使用しても1回のRSA演算で平均1500回程度の多重精度乗算と剰余算とを行わなければならない。特に剰余計算は、近似法、剰余テーブル方式、モンゴメリのアルゴリズム等、多くの高速化手法が提案されている。このような、RSA暗号に代表される公開鍵暗号系の多くで利用される、べき乗剰余アルゴリズムを高速に処理するためには、1回あたりの剰余アルゴリズムの高速化が要求される。

【0013】この剰余演算の高速化を実現する一方法であるモンゴメリのアルゴリズムについて説明する。

(モンゴメリのアルゴリズム) モンゴメリのアルゴリズムは、剰余の法 N ($N > 1$) と、剰余の法 N と互いに素である基数 R ($R > N$) とを用いると、被剰余数 T から $TR^{-1} \bmod N$ の計算が基数 R による除算のみで行えることを利用して、 N による除算を用いることなく剰余計算を行うアルゴリズムである。ここで、 N, N', R, R^{-1} 及び T は整数であり、被剰余数 T は $0 \leq T < R \cdot N$ 、 R^{-1} は剰余の法 N の上での基数 R の逆数であり、 $R \cdot R^{-1} - N \cdot N' = 1$ ($0 \leq R^{-1} < N, 0 \leq N' < R$) の関係を満たす。

【0014】更に、この基数 R に2のべき乗数を使用した場合、基数 R による除算をシフト操作に置き換えることができるため、 $T \rightarrow TR^{-1} \bmod N$ の計算の高速処理が可能となる。次に、アルゴリズム1として、 $T \rightarrow TR^{-1} \bmod N$ のアルゴリズム REDC (T) を示す。但し、アルゴリズム1において $(T + m \cdot N) / R$ は必ず割り切れることが証明されている。

【0015】(アルゴリズム1) $T \rightarrow TR^{-1} \bmod N$ のアルゴリズム $Y = \text{REDC}(T)$ は次のようにあらわされる。

$$M = (T \bmod R) \cdot N' \bmod R$$

$$Y = (T + M \cdot N) / R$$
 if $Y \geq N$ then $Y = Y - N$
 $Y < N$ then return Y

05 【0016】1回のREDCでは、剰余 $T \bmod N$ ではなく $TR^{-1} \bmod N$ が求められるだけである。よって、剰余 $T \bmod N$ を求めるためには、次に示すように REDC (T) と予め求めておいた $R^2 \bmod N$ との積で、再び REDC を行えば良い。

$$\begin{aligned} 10 \text{ REDC}(\text{REDC}(T) \cdot (R^2 \bmod N)) \\ = (TR^{-1} \bmod N) \cdot (R^2 \bmod N) \cdot R^{-1} \bmod N \\ = TR^{-1} \cdot R^2 \cdot R^{-1} \bmod N \\ = T \bmod N \end{aligned}$$

このようにして、剰余 $T \bmod N$ を求めることができる。

15 【0017】(REDCの多重精度計算への拡張) 次に、剰余の法 N または基数 R が多倍長即ち多重精度である場合について、REDCのアルゴリズムを拡張する。剰余の法 N 、基数 R が多重精度である場合、REDCの ($T \bmod R$) $\cdot N'$ 及び $M \cdot N$ の計算は、多重精度 \times 多重精度の処理となり、汎用の計算機では非常に大きな処理量と処理時間とが必要となる。そこで、この部分を多重精度 \times 単精度の処理で行えるように拡張したアルゴリズム2を示す。

【0018】(アルゴリズム2) REDCを多重精度へ拡張したアルゴリズムは次に示すようになる。被剰余数 T 、パラメータ N' 、出力用変数 Y が何れも r 進数で、

$$T = (t_{2^r-1}, t_{2^r-2}, \dots, t_0)_r,$$

$$N' = (n'_{r-1}, n'_{r-2}, \dots, n'_0)_r,$$

$$Y = (y_g, y_{g-1}, \dots, y_0)_r,$$

$$\begin{aligned} 25 \text{ } R &= r^g, \\ \text{ } r &= 2^k \end{aligned}$$

とあらわされる場合、次に示す $j = 0 \sim g-1$ の繰り返し処理により $TR^{-1} \bmod N$ を多重精度 \times 単精度として求めることができる。ここで単精度とは r 進数1桁のこととし、同じ文字を使用した場合、基本的に大文字を多重精度、小文字を単精度、小文字の添字を多重精度での桁の位置とする。

【0019】 $Y = T$
 for $j = 0$ to $g-1$
 40 $m = y_0 \cdot n'_0 \bmod r$
 $Y = Y + m \cdot N$
 $Y = Y / r$
 next
 if $Y \geq N$ then $Y = Y - N$
 45 $Y < N$ then return Y

このようにして得られる $TR^{-1} \bmod N$ と、上述したように予め求めておいた $R^2 \bmod N$ との積で再び REDC を行うことにより、 $TR^{-1} \bmod N$ を求めることができる。

【0020】(REDCの多重精度乗算剰余への拡張)

50 次に、REDCのアルゴリズムを乗算剰余演算に拡張す

る。上記のアルゴリズムにおいて、入力 T は $0 \leq T < R \cdot N$ を満たす値であるが、実際の RSA 演算では、入力 T が整数 A , B ($0 \leq A, B < N$) の乗算結果であることが多い。その場合、整数 A , B の乗算も多重精度整数演算であるため、多重精度拡張 REDC と同様の繰り返し計算が行われる。この場合、乗算と REDC とを別々に繰り返し計算すると、繰り返し計算制御によるロスが 2 倍になってしまう。そこで、乗算と REDC とを同一の繰り返しループで行えるように拡張したアルゴリズム 3 を示す。

【0021】(アルゴリズム 3) REDC を多重精度乗算剰余へ拡張したアルゴリズム REDC ($A \times B$) は次に示すようになる。乗算する 2 数 A , B , パラメータ N' , 出力用変数 Y が何れも r 進数で、

$$A = (a_{g-1}, a_{g-2}, \dots, a_0)_r,$$

$$B = (b_{g-1}, b_{g-2}, \dots, b_0)_r,$$

$$N' = (n'_{g-1}, n'_{g-2}, \dots, n'_0)_r,$$

$$Y = (y_g, y_{g-1}, \dots, y_0)_r,$$

$$R = r^g,$$

$$r = 2^k$$

とあらわされる場合、次に示す $j = 0 \sim g-1$ の繰り返し処理により、 $ABR^{-1} \bmod N$ を多重精度 \times 単精度の計算として求めることができる。

【0022】 $Y = 0$

for $j = 0$ to $g-1$

$$Y = Y + A \cdot b_j$$

$$m = y_0 \cdot n'_0 \bmod r$$

$$Y = Y + m \cdot N$$

$$Y = Y / r$$

next

if $Y \geq N$ then $Y = Y - N$

$Y < N$ then return Y

このようにして得られる $ABR^{-1} \bmod N$ と、上述したよ

$$Y = 0$$

for $j = 0$ to $g-1$

$$(tmp2, tmp1)_r = y_0 + a_i \cdot b_j$$

$$m = tmp1 \cdot n'_0 \bmod r$$

$$(tmp4, tmp1)_r = tmp1 + m \cdot n_0$$

$$(c_1, c_0)_r = tmp2 + tmp4$$

for $i = 0$ to $g-1$

$$(tmp3, tmp2, tmp1)_r = y_i + (c_1, c_0)_r + a_i \cdot b_j$$

$$(tmp4, y_{i-1})_r = tmp1 + m \cdot n_i$$

$$(c_1, c_0)_r = tmp4 + (tmp3, tmp2)_r$$

next i

$$(c_1, c_0)_r = (c_1, c_0)_r + y_g$$

$$y_{g-1} = c_0$$

$$y_g = c_1$$

next j

if $Y \geq N$ then $Y = Y - N$

$Y < N$ then return Y

うに予め求めておいた $R^2 \bmod N$ との積で再び REDC を行うことにより、 $ABR^{-1} \bmod N$ を求めることができる。

【0023】(REDC の単精度 \times 単精度処理への拡張)

- 05 アルゴリズム 3 では、多重精度のモンゴメリ乗算剰余を多重精度 \times 単精度で実現可能としているが、この多重精度 \times 単精度の計算部分をさらに単精度 \times 単精度の計算を組み合わせて行えるよう拡張する。この場合、 $A \times b_i$ の計算部分と $m \times N$ の計算部分とが繰り返し計算となり、上述の場合と同様に 2 つの乗算を別々に繰り返し計算すると、繰り返し計算制御によるロスが 2 倍になってしまう。そこで、2 つの乗算を同一の繰り返しループで行えるようにすれば、ロスの低減が可能である。2 つの乗算を同一の繰り返しループで行えるように拡張したアルゴリズム 4 を示す。

【0024】(アルゴリズム 4) REDC を単精度 \times 単精度へ拡張したアルゴリズム REDC ($A \times B$) は次に示すようになる。乗算する 2 数 A , B , パラメータ N' , 出力用変数 Y , キャリー変数 C が何れも r 進数

20 で、

$$A = (a_{g-1}, a_{g-2}, \dots, a_0)_r,$$

$$B = (b_{g-1}, b_{g-2}, \dots, b_0)_r,$$

$$N' = (n'_{g-1}, n'_{g-2}, \dots, n'_0)_r,$$

$$Y = (y_g, y_{g-1}, \dots, y_0)_r,$$

25 $C = (c_1, c_0)_r,$

$$R = r^g,$$

$$r = 2^k$$

とあらわされ、 r 進 1 桁の一時変数 $tmp1$, $tmp2$, $tmp3$, $tmp4$ とする場合、次に示す i, j の繰り返し処理により

- 30 $ABR^{-1} \bmod N$ を単精度 \times 単精度の計算で求めることができる。

【0025】

コア前処理

コア処理

コア後処理

補正処理

【0026】ここで、 $()_r$ は、括弧内の r 進数1桁の変数を多重精度として扱うことを示している。 $tmp3$ 、 c_1 は r 進数1桁で表現しているが、内容は1ビットの値である。出力用変数 Y について、計算に使用する値が y_1 のとき、出力が y_{i-1} に格納されるのは、アルゴリズム3における $Y=Y/r$ の機能をこれにより実現しているためである。また、便宜上、外側のループを j ループ、内側のループを i ループと呼び、 j ループの始めから i ループまでをコア前処理、 i ループ内の処理をコア処理、 i ループの終わりから j ループの終わりまでをコア後処理と呼ぶこととする。

【0027】図11は、上述したアルゴリズム4のコア処理を実行する乗算剰余計算装置の構成図である。図11に示す乗算剰余計算装置は、内部で乗算及び加算を行う α 積和回路51及び β 積和回路52と、乗算する一方の数 A ： $(a_{r-1}, a_{r-2}, \dots, a_0)$ を保持するAレジスタ53と、乗算する一方の数 B ： $(b_{r-1}, b_{r-2}, \dots, b_0)$ を保持するBレジスタ54と、剰余の法 N ： $(n_{r-1}, n_{r-2}, \dots, n_0)$ を保持するNレジスタ55と、 β 積和回路52の出力の下位 k ビットを格納するYレジスタ56と、モンゴメリのパラータ m を保持するmレジスタ57と、 α 積和回路51の出力の上位 $(k+1)$ ビット及び β 積和回路52の出力の上位 k ビットを加算するキャリ計算部としての加算回路58と、加算回路58の加算結果を格納するCレジスタ59と、 j の値と0とを比較して出力を選択する選択回路60とを有する。

【0028】また、 α 積和回路51、 β 積和回路52の内部構成を図12(a)、(b)に夫々示す。 α 積和回路51は、Aレジスタ53及びBレジスタ54からの出力を乗算する k ビット乗算器511と、 k ビット乗算器511の出力及び選択回路60(Yレジスタ56)の出力を加算する $2k$ ビット加算器512と、 $2k$ ビット加算器512の出力及びCレジスタ59の出力を加算する $2k+1$ ビット加算器513とを有する。 β 積和回路52は、Nレジスタ55及びmレジスタ57からの出力を乗算する k ビット乗算器521と、 k ビット乗算器521の出力及び α 積和回路51からの下位 k ビットの出力を加算する $2k$ ビット加算器522とを有する。

【0029】図13は、アルゴリズム4のコア処理の内容を示す説明図である。 α 積和回路51内にて、Aレジスタ53の出力 a_1 (k ビット)とBレジスタ54の出力 b_j (k ビット)とを乗算し、その乗算結果($2k$ ビット)に、選択回路60(Yレジスタ56)の出力(k ビット)とCレジスタ59の出力($k+1$ ビット)とを加算する。なお、選択回路60は、 j の値と0とを比較し、 j の値が0である場合には α 積和回路51へ0を出力し、 j の値が0でない場合にはYレジスタ56の格納値 y_1 を α 積和回路51へ出力する。 α 積和回路51は、その演算結果($2k+1$ ビット)の上位 $(k+1)$ ビットを加算回路58へ出力し、その下位 k ビットを β 積和回路52へ出力する。

【0030】 β 積和回路52内にて、Nレジスタ55の出力 n_1 (k ビット)とmレジスタ57の出力(k ビット)とを乗算し、その乗算結果($2k$ ビット)に α 積和回路51からの下位 k ビット出力を加算する。 β 積和回路52は、その演算結果($2k$ ビット)の上位 k ビットを加算回路58へ出力し、その下位 k ビットをYレジスタ56へ出力する。Yレジスタ56は、その k ビットのデータを値 y_{i-1} として格納する。

【0031】加算回路58は、 α 積和回路51からの出力($k+1$ ビット)と β 積和回路52からの出力(k ビット)とを加算し、その加算結果($k+1$ ビット)をCレジスタ59へ出力する。Cレジスタ59は、これを格納する。

【0032】

【発明が解決しようとする課題】上述したような従来の乗算剰余計算装置の構成では、2つの乗算を同一の繰返しループで行えるが、Cレジスタ59から出力される次ループへのキャリが $(k+1)$ ビットとなり、そのため α 積和回路51の出力が $(2k+1)$ ビットとなってしまっており、1つ目の積和回路(α 積和回路51)の構成を単純化できないという問題がある。

【0033】また、キャリ計算部である加算回路58において、次回演算用のキャリが2つの α 積和回路51及び β 積和回路52での演算結果に基づいて計算されるため、2つ目の積和回路(β 積和回路52)での演算が終了しなければ、次回の1つ目の積和回路(α 積和回路51)での演算を行うことができず、処理能率が悪いという問題がある。

【0034】本発明は斯かる事情に鑑みてなされたものであり、1つ目の積和回路への次回演算用のキャリを k ビットにすることにより、積和回路の構成を単純化することができ、また、これによりDSP(Digital Signal Processor)、マイクロコントローラ等の上のソフトウェアとして実現が容易になるモンゴメリ法による乗算剰余計算装置を提供することを目的とする。

【0035】本発明の他の目的は、キャリーの伝搬を各積和回路でのループ内に押さえることにより、2つの積和回路での演算を独立して行え、つまり、2つ目の積和回路での積和を行っている間に1つ目の積和回路で次回の積和を行え、これによりパイプライン処理が可能になるモンゴメリ法による乗算剰余計算装置を提供することにある。

【0036】

【課題を解決するための手段】請求項1に係るモンゴメリ法による乗算剰余計算装置(第1発明)は、モンゴメリのアルゴリズムを用いて乗算剰余計算を行う装置において、積和演算を行いその演算結果を上位 k ビットと下位 k ビットとに分けて出力する第1積和回路と、積和演算を行いその演算結果を上位 k ビットと下位 k ビットとに分けて出力する第2積和回路と、加算演算を行いその

演算結果を上位 1 ビットと下位 k ビットとに分けて出力する加算回路と、乗算される 2 数を保持する第 1 及び第 2 レジスタと、前記第 2 積和回路の下位 k ビット出力を保持し、前記第 2 積和回路のその次の回の下位 k ビット出力を格納する第 3 レジスタと、前記加算回路の下位 k ビット出力を保持し、前記加算回路のその次の回の下位 k ビット出力を格納する第 4 レジスタと、剰余の法を保持する第 5 レジスタと、モンゴメリのアルゴリズムにおけるパラメータの値を保持する第 6 レジスタと、前記加算回路の上位 1 ビット出力を保持し、前記加算回路のその次の回の下位 1 ビット出力を格納する第 7 レジスタとを備え、前記第 1 積和回路は、前記第 1 及び第 2 レジスタに保持された 2 数の所定ビットの値を乗算し、その乗算結果に前記第 3 レジスタに保持された数の所定ビットの値及び前記第 4 レジスタに保持された値を加算する演算を行い、前記第 2 積和回路は、前記第 5 レジスタに保持された数の所定ビットの値と前記第 6 レジスタに保持された値とを乗算し、その乗算結果に前記第 1 積和回路の下位 k ビット出力を加算する演算を行い、前記加算回路は、前記第 1 積和回路の上位 k ビット出力と前記第 2 積和回路の上位 k ビット出力と前記第 7 レジスタに保持された値とを加算する演算を行うように構成したことを特徴とする。

【0037】請求項 2 に係るモンゴメリ法による乗算剰余計算装置は、請求項 1 において、前記第 1 積和回路は、前記第 1 及び第 2 レジスタに保持された 2 数の所定ビットの値を乗算し、その乗算結果に前記第 3 レジスタに保持された数の所定ビットの値を加算し、その加算結果に前記第 4 レジスタに保持された値を加算するように構成したことを特徴とする。

【0038】請求項 3 に係るモンゴメリ法による乗算剰余計算装置は、請求項 1 において、前記第 1 積和回路は、前記第 1 及び第 2 レジスタに保持された 2 数の所定ビットの値を乗算し、その乗算結果に前記第 4 レジスタに保持された値を加算し、その加算結果に前記第 3 レジスタに保持された数の所定ビットの値を加算するように構成したことを特徴とする。

【0039】請求項 4 に係るモンゴメリ法による乗算剰余計算装置（第 2 発明）は、モンゴメリのアルゴリズムを用いて乗算剰余計算を行う装置において、積和演算を行いその演算結果を上位 k ビットと下位 k ビットとに分けて出力する第 1 積和回路と、積和演算を行いその演算結果を上位 k ビットと下位 k ビットとに分けて出力する第 2 積和回路と、乗算される 2 数を保持する第 1 及び第 2 レジスタと、前記第 2 積和回路の下位 k ビット出力を保持し、前記第 2 積和回路のその次の回の下位 k ビット出力を格納する第 3 レジスタと、前記第 1 積和回路の上位 k ビット出力を保持し、前記第 1 積和回路のその次の回の下位 k ビット出力を格納する第 4 レジスタと、剰余の法を保持する第 5 レジスタと、モンゴメリのアルゴリ

ズムにおけるパラメータの値を保持する第 6 レジスタと、前記第 2 積和回路の上位 k ビット出力を保持し、前記第 2 積和回路のその次の回の下位 k ビット出力を格納する第 7 レジスタとを備え、前記第 1 積和回路は、前記第 1 及び第 2 レジスタに保持された 2 数の所定ビットの値を乗算し、その乗算結果に前記第 3 レジスタに保持された数の所定ビットの値及び前記第 4 レジスタに保持された値を加算する演算を行い、前記第 2 積和回路は、前記第 5 レジスタに保持された数の所定ビットの値と前記第 6 レジスタに保持された値とを乗算し、その乗算結果に前記第 1 積和回路の下位 k ビット出力及び前記第 7 レジスタに保持された値を加算する演算を行うように構成したことを特徴とする。

【0040】請求項 5 に係るモンゴメリ法による乗算剰余計算装置は、請求項 4 において、前記第 1 積和回路は、前記第 1 及び第 2 レジスタに保持された 2 数の所定ビットの値を乗算し、その乗算結果に前記第 3 レジスタに保持された数の所定ビットの値を加算し、その加算結果に前記第 4 レジスタに保持された値を加算するように構成したことを特徴とする。

【0041】請求項 6 に係るモンゴメリ法による乗算剰余計算装置は、請求項 4 において、前記第 1 積和回路は、前記第 1 及び第 2 レジスタに保持された 2 数の所定ビットの値を乗算し、その乗算結果に前記第 4 レジスタに保持された値を加算し、その加算結果に前記第 3 レジスタに保持された数の所定ビットの値を加算するように構成したことを特徴とする。

【0042】請求項 7 に係るモンゴメリ法による乗算剰余計算装置は、請求項 4 において、前記第 2 積和回路は、前記第 5 レジスタに保持された数の所定ビットの値と前記第 6 レジスタに保持された値とを乗算し、その乗算結果に前記第 1 積和回路の下位 k ビット出力を加算し、その加算結果に前記第 7 レジスタに保持された値を加算するように構成したことを特徴とする。

【0043】請求項 8 に係るモンゴメリ法による乗算剰余計算装置は、請求項 4 において、前記第 2 積和回路は、前記第 5 レジスタに保持された数の所定ビットの値と前記第 6 レジスタに保持された値とを乗算し、その乗算結果に前記第 7 レジスタに保持された値を加算し、その加算結果に前記第 1 積和回路の下位 k ビット出力を加算するように構成したことを特徴とする。

【0044】請求項 9 に係るモンゴメリ法による乗算剰余計算装置は、請求項 4～8 の何れかにおいて、前記第 2 積和回路による演算中に、前記第 1 積和回路によりその次の回の演算を行うように構成したことを特徴とする。

【0045】請求項 10 に係るモンゴメリ法による乗算剰余計算装置は、暗号化／復号化のためのモンゴメリ法による乗算剰余計算を行う装置において、乗算演算を行う第 1 積和手段と、モンゴメリ剰余演算を行う第 2 積

和手段とを備え、前記第1積和手段の出力を前記第2積和手段の入力とする構成を有し、前記第2積和手段が演算を行う間、前記第1積和手段で次の回の演算を行うように、パイプライン処理すべく構成したことを特徴とする。

【0046】請求項11に係るモンゴメリ法による乗算剰余計算装置は、請求項10において、前記第1積和手段及び第2積和手段にあって、各自身の上位出力を各自身の次の回のキャリア入力とするようにしたことを特徴とする。

【0047】請求項12に係るモンゴメリ法による乗算剰余計算装置は、請求項10において、前記第1積和手段及び第2積和手段における演算量が等しいことを特徴とする。

【0048】請求項13に係るモンゴメリ法による乗算剰余計算装置は、請求項10において、前記第1積和手段及び第2積和手段は、2つのkビットの数を乗算する手段と、その乗算結果に2つのkビットの数を加算する手段とを有することを特徴とする。

【0049】第1発明の乗算剰余計算装置では、従来例において問題となっている(k+1)ビットのキャリアを上位1ビットと下位kビットとに分離し、その下位kビットは1つのキャリアとして従来例と同様に1つ目の積和回路に戻し、その上位1ビットはもう1つのキャリアとして、キャリア計算用の加算回路に戻す。この構成をとることにより、1つ目の積和回路の構成の単純化を図れる。

【0050】第2発明の乗算剰余計算装置では、2つの積和回路から出力される上位データを加算するのではなく、各積和回路におけるキャリア用のレジスタを各別に

設けて自身の積和回路にキャリアを戻す構成とすることにより、第1発明と同様に1つ目の積和回路の構成の単純化を図れると共に、更に各積和回路のキャリア処理が閉じているので、2つ目の積和回路の動作中に、1つ目の積和回路の次の動作が可能となる。

【0051】

【発明の実施の形態】以下、本発明をその実施の形態を示す図面を参照して具体的に説明する。

(第1発明) 1つ目の積和回路へのキャリアをkビットにした第1発明について説明する。第1発明では、1つ目の積和回路の出力と2つ目の積和回路の出力との加算結果である(k+1)ビットのキャリアを上位1ビットと下位kビットとに分離し、その下位kビットはキャリア変数c1として1つ目の積和回路に戻し、その上位1ビットはもう1つのキャリア変数c2として、キャリア計算用の加算回路に戻す。この場合のアルゴリズム5を以下に示す。

【0052】(アルゴリズム5) 乗算する2数A, B, パラメータN', 出力用変数Yが何れもr進数で、

20 $A = (a_{r-1}, a_{r-2}, \dots, a_0)_r,$
 $B = (b_{r-1}, b_{r-2}, \dots, b_0)_r,$
 $N' = (n'_{r-1}, n'_{r-2}, \dots, n'_0)_r,$
 $Y = (y_r, y_{r-1}, \dots, y_0)_r,$
 $R = r^k,$
 25 $r = 2^k$

とあらわされ、r進1桁の一時変数tmp1, tmp2, tmp4, キャリア変数c1, c2とする場合、次に示すi, jの繰返し処理により $ABR^{-1} \bmod N$ を単精度×単精度の計算として求めることができる。

30 【0053】

```

Y=0
for j=0 to g-1
  (tmp2, tmp1)r = y0 + a1 · bj
  m = tmp1 · n' 0 mod r
  (tmp4, tmp1)r = tmp1 + m · n0
  (c2, c1)r = tmp2 + tmp4
  for i=1 to g-1
    (tmp2, tmp1)r = y1 + c1 + a1 · bj
    (tmp4, yi-1)r = tmp1 + m · n1
    (c2, c1)r = tmp4 + tmp2 + c2
  next i
  (c2, c1)r = (c2, c1)r + yr
  yr-1 = c1
  yr = c2
next j
if Y ≥ N then Y = Y - N
Y < N then return Y

```

.....

コア前処理

.....

コア処理

.....

コア後処理

.....

補正処理

ここで、()_rは、括弧内のr進数1桁の変数を多重精度として扱うことを示している。またキャリア変数c2はr進数1桁で表現しているが、内容は1ビットの値

である。

【0054】図1は、上述したアルゴリズム5のコア処理を実行する乗算剰余計算装置の構成図である。図1に

示す乗算剰余計算装置は、内部で乗算及び加算を行う第1積和回路1及び第2積和回路2と、乗算する一方の数A: $(a_{r-1}, a_{r-2}, \dots, a_0)$ を保持する第1レジスタとしてのAレジスタ3と、乗算する一方の数B:

$(b_{r-1}, b_{r-2}, \dots, b_0)$ を保持する第2レジスタとしてのBレジスタ4と、第2積和回路2の前の下位kビット出力を保持し次回の下位kビット出力を格納する第3レジスタとしてのYレジスタ5と、キャリー変数c1を保持する第4レジスタとしてのc1レジスタ6と、剰余の法N: $(n_{r-1}, n_{r-2}, \dots, n_0)$ を保持する第5レジスタとしてのNレジスタ7と、モンゴメリアルゴリズムにおけるパラータmを保持する第6レジスタとしてのmレジスタ8と、キャリー変数c2を保持する第7レジスタとしてのc2レジスタ9と、第1積和回路1の上位kビット出力、第2積和回路2の上位kビット出力及びc2レジスタ9の出力を加算するキャリー計算部としての加算回路10と、jの値と0とを比較してその出力を選択する選択回路11とを有する。

【0055】また、第1積和回路1、第2積和回路2の内部構成を図2(a)、(b)に夫々示す。第1積和回路1は、kビット乗算器101と2kビット加算器102と2kビット加算器103とを有する。kビット乗算器101は、Aレジスタ3及びBレジスタ4からの出力を乗算し、2kビット加算器102は、kビット乗算器101の出力と選択回路11(Yレジスタ5)の出力とを加算し、2kビット加算器103は、2kビット加算器102の出力とc1レジスタ6の出力とを加算する。なお、図2(a)に示す構成例では、乗算結果に選択回路11(Yレジスタ5)の出力を先に加算し、その後にc1レジスタ6の出力を加算するようになっているが、これとは逆に、先にc1レジスタ6の出力、その後に選択回路11(Yレジスタ5)の出力を加算するように構成しても良い。

【0056】第2積和回路2は、Nレジスタ7及びmレジスタ8からの出力を乗算するkビット乗算器201と、kビット乗算器201の出力及び第1積和回路1からの下位kビットの出力を加算する2kビット加算器202とを有する。

【0057】図3は、アルゴリズム5のコア処理の内容を示す説明図である。第1積和回路1内にて、Aレジスタ3の出力 a_i (kビット)とBレジスタ4の出力 b_j (kビット)とを乗算し、その乗算結果(2kビット)

に、選択回路11(Yレジスタ5)の出力(kビット)とc1レジスタ6の出力(kビット)とを加算する。なお、選択回路11は、jの値と0とを比較し、jの値が0である場合には第1積和回路1へ0を出力し、jの値が0でない場合にはYレジスタ5の格納値 y_i を第1積和回路1へ出力する。第1積和回路1は、その演算結果(2kビット)の上位kビットを加算回路10へ出力し、その下位kビットを第2積和回路2へ出力する。

【0058】第2積和回路2内にて、Nレジスタ7の出力 n_i (kビット)とmレジスタ8の出力(kビット)とを乗算し、その乗算結果(2kビット)に、第1積和回路1からの出力下位kビットを加算する。第2積和回路2は、その演算結果(2kビット)の上位kビットを加算回路10へ出力し、その下位kビットをYレジスタ5へ出力する。Yレジスタ5は、そのkビットのデータを値 y_{i-1} として格納する。

【0059】加算回路10は、第1積和回路1からの出力(kビット)と第2積和回路2からの出力(kビット)とc2レジスタ9からの出力(1ビット)とを加算する。そして、次回の演算用として、その加算結果(k+1ビット)の上位1ビットをc2レジスタ9へ、その下位kビットをc1レジスタ6へ夫々出力する。各c1レジスタ6、c2レジスタ9は、これを格納する。

【0060】(第2発明) キャリー用のレジスタを各積和回路毎に設けた第2発明について説明する。第2発明では、各積和回路におけるキャリー用のレジスタを各別に設けて自身の積和回路にキャリー変数を戻す構成とする。このアルゴリズム6を以下に示す。

【0061】(アルゴリズム6) 乗算する2数A, B, パラメータ N' , 出力用変数Yが何れもr進数で、

$$A = (a_{r-1}, a_{r-2}, \dots, a_0)_r,$$

$$B = (b_{r-1}, b_{r-2}, \dots, b_0)_r,$$

$$N' = (n'_{r-1}, n'_{r-2}, \dots, n'_0)_r,$$

$$Y = (y_r, y_{r-1}, \dots, y_0)_r,$$

$$R = r^k,$$

$$r = 2^k$$

とあらわされ、r進1桁の一時変数tmpl, キャリー変数c3, c4とする場合、次に示すi, jの繰り返し処理により $ABR^{-1} \bmod N$ を単精度×単精度の計算として求めることができる。

【0062】

```

Y = 0
for j = 0 to g-1
    (c3, tmp1)r = y0 + ai · bj
    m = tmp1 · n' 0 mod r
    (c4, tmp1)r = tmp1 + m · n0
for i = 1 to g-1
    (c3, tmp1)r = y1 + c3 + ai · bj
    (c4, yi-1)r = tmp1 + m · ni
next i

```

コア前処理

コア処理

```

(c4, c3)r = c3 + c4 + yr
yr-1 = c3
yr = c4
next j
if Y ≥ N then Y = Y - N
Y < N then return Y

```

コア後処理

補正処理

【0063】図4は、上述したアルゴリズム6のコア処理を実行する乗算剰余計算装置の構成図である。図4に示す乗算剰余計算装置は、内部で乗算及び加算を行う第3積和回路21及び第4積和回路22と、図1に示すものと同様の第1レジスタとしてのAレジスタ3、第2レジスタとしてのBレジスタ4、第3レジスタとしてのYレジスタ5、第5レジスタとしてのNレジスタ7、第6レジスタとしてのmレジスタ8及び選択回路11と、キャリー変数c3を保持する第4レジスタとしてのc3レジスタ26と、キャリー変数c4を保持する第7レジスタとしてのc4レジスタ29とを有する。

【0064】なお、第3積和回路21及び第4積和回路22の内部構成は、図2(a)に示す第1積和回路1の内部構成と同じであり、各積和回路21及び22は、kビット乗算器101と2kビット加算器102と2kビット加算器103とから構成されている。

【0065】第3積和回路21のkビット乗算器101は、Aレジスタ3及びBレジスタ4からの出力を乗算し、2kビット加算器102は、kビット乗算器101の出力と選択回路11(Yレジスタ5)の出力とを加算し、2kビット加算器103は、2kビット加算器102の出力とc3レジスタ26の出力とを加算する。なお、図2(a)に示す構成例では、乗算結果に選択回路11(Yレジスタ5)の出力を先に加算し、その後にc3レジスタ26の出力を加算するようになっているが、これとは逆に、先にc3レジスタ26の出力、その後に選択回路11(Yレジスタ5)の出力を加算するように構成しても良い。

【0066】一方、第4積和回路22のkビット乗算器101は、Nレジスタ7及びmレジスタ8からの出力を乗算し、2kビット加算器102は、kビット乗算器101の出力と第3積和回路21からの下位kビットの出力とを加算し、2kビット加算器103は、2kビット加算器102の出力とc4レジスタ29の出力とを加算する。なお、図2

(a)に示す構成例では、乗算結果に第3積和回路21からの下位kビットの出力を先に加算し、その後にc4レジスタ29の出力を加算するようになっているが、これとは逆に、先にc4レジスタ29の出力、その後に第3積和回路21からの下位kビットの出力を加算するように構成しても良い。

【0067】図5は、アルゴリズム6のコア処理の内容を示す説明図である。第3積和回路21内にて、Aレジスタ3の出力a_i(kビット)とBレジスタ4の出力b_j(kビット)とを乗算し、その乗算結果(2kビット)に、選択回路11(Yレジスタ5)の出力(kビット)と

c3レジスタ26の出力(kビット)とを加算する。なお、選択回路11は、jの値と0とを比較し、jの値が0である場合には第3積和回路21へ0を出力し、jの値が0でない場合には第3Yレジスタ5の格納値y_jを積和回路21へ出力する。第3積和回路21は、その演算結果(2kビット)の上位kビットをc3レジスタ26へ出力し、その下位kビットを第4積和回路22へ出力する。c3レジスタ26は、このkビットを次の演算用のキャリー変数として格納する。

【0068】第4積和回路22内にて、Nレジスタ7の出力n_i(kビット)とmレジスタ8の出力m(kビット)とを乗算し、その乗算結果(2kビット)に、第3積和回路21からの下位kビット出力を加算する。第4積和回路22は、その演算結果(2kビット)の上位kビットをc4レジスタ29へ出力し、その下位kビットをYレジスタ5へ出力する。c4レジスタ29は、このkビットを次の演算用のキャリー変数として格納する。また、Yレジスタ5は、そのkビットのデータを値y_{i-1}として格納する。

【0069】図6は、モンゴメリ法による乗算剰余処理の一例を示すフローチャートである。このフローチャートにおいて、jループが(アルゴリズム3)のループ処理に当たる。jループの内側では、A×b_j及びm×Nの多重精度×単精度の部分乗算を行っている。iループは、A×b_j及びm×Nの多重精度×単精度の計算を単精度×単精度の部分乗算で行っている部分である。iループの内部ではa_i×b_jとm×n_iとの部分乗算を行っている。

【0070】次に、第2発明における更なる具体例について説明する。以下の例では、N、A、Bのビット長を1024ビット、g=32、処理単位k=32、R=2¹⁰²⁴、r=2³²とする。

【0071】(コア前処理)図7は、コア前処理を行う構成の一例を示す図である。31はモンゴメリ計算用のパラメータn'を保持するレジスタ、32は第3積和回路21の出力とレジスタ31の出力とを乗算する乗算回路である。

【0072】このコア前処理では、コア処理で使用するc3レジスタ26、c4レジスタ29及びmレジスタ8の初期化を行っている。第3積和回路21は、まず、Aレジスタ3、Bレジスタ4からの入力a₀、b_jを乗算し、その乗算結果とYレジスタ5からの入力y_iとを加算する。なお、コア処理と同じ積和回路を使用する場合は、更にその結果と0とを加算する。そして、結果の上位32

ビットをc3 レジスタ26に格納し、下位32ビットを第3積和回路21とパラメータmを計算するための乗算回路32とへ出力する。乗算回路32は、第3積和回路21の出力とレジスタ31の出力 n' とを乗算し、その乗算結果の下位32ビットをmレジスタへに出力する。

【0073】第4積和回路22は、Nレジスタ7からの入力 n_0 とmレジスタ8の値とを乗算し、その乗算結果と第3積和回路21からの出力とを加算する。なお、コア処理と同じ積和回路を使用する場合は、更にその結果と0とを加算する。そして、結果の上位32ビットをc4 レジスタ29に格納する。下位32ビットは使用しない。

【0074】(コア処理) 図8は、iループ内部処理であるコア処理を行う構成の一例を示す図である。Yレジスタ5は前回の処理結果の保持及び今回の処理結果の出力用レジスタである。選択回路11は、(アルゴリズム3)で $Y=0$ の処理に相当するものである。

【0075】第3積和回路21は、まず、Aレジスタ3、Bレジスタ4からの入力 a_i 、 b_j を乗算し、その乗算結果とYレジスタ5からの入力 y_i とを加算し、更にその加算結果とc3 レジスタ26の値とを加算する。そして、結果の上位32ビットをc3レジスタ26に格納し、下位32ビットを第4積和回路22へ出力する。

【0076】第4積和回路22は、まず、Nレジスタ7からの入力 n_i とmレジスタ8の値とを乗算し、その乗算結果と第3積和回路21からの出力とを加算し、更にその加算結果とc4 レジスタ29の値とを加算する。そして、結果の上位32ビットをc4 レジスタ29に格納し、下位32ビットをYレジスタ5の y_{i+1} に格納する。(アルゴリズム3)の $Y=Y/r$ の処理は、i回目の計算結果を y_{i+1} に格納することで実現している。

【0077】(コア後処理) 図9は、コア後処理を行う構成の一例を示す図である。33はc3 レジスタ26の出力とc4 レジスタ29の出力と選択回路11の出力とを加算する加算回路、34は加算回路33からのキャリー出力を0、1と比較し、0であれば0を、1であれば1を、Yレジスタ5へ出力する選択回路である。

【0078】このコア後処理では、コア処理終了後のキャリー変数c3、c4の値の処理を行っている。c3 レジスタ26、c4 レジスタ29の値及びYレジスタ5からの入力 y_{32} を加算回路33に入力し、その加算結果をYレジスタ5の y_{31} に出力し、キャリーを処理単位である32ビットの値に変換してYレジスタ5の y_{32} に出力する。ここで、出力からもわかるように、 y_{32} の値はYレジスタ5では32ビットとして扱われているが、実際は1ビットの値であるので、加算結果は32ビット+キャリーの範囲で収まる。

【0079】(積和回路の構成) 図10は、上述の構成例で用いた積和回路の構成の一例を示す図である。ここでは、全ての処理単位を32ビットになるように構成している。積和回路は、1個の32ビット乗算器41と、4個の32

ビット加算器42、43、44、45とを有する。

【0080】A、Bの入力値は32ビット乗算器41で乗算され、上位32ビットと下位32ビットとの2つで出力される。32ビット加算器43は、32ビット乗算器41の出力下位32ビットと入力Rの値とを加算し、その加算結果の出力32ビットを32ビット加算器45へ、キャリーを32ビット加算器42へそれぞれ出力する。32ビット加算器42は、32ビット乗算器41の出力上位32ビットと32ビット加算器43のキャリー出力とを加算し、その加算結果の出力32ビットを32ビット加算器44へ出力する。この加算ではキャリーが発生しないことが理論的に証明されている。

【0081】32ビット加算器45は、32ビット加算器43の出力と入力Cの値とを加算し、その加算結果の出力32ビットは積和回路のL出力(下位32ビット)となり、キャリーは32ビット加算器44へ出力される。32ビット加算器44は、32ビット加算器42の出力と32ビット加算器45のキャリー出力とを加算し、その加算結果の出力32ビットは積和回路のH出力(上位32ビット)となる。この加算ではキャリーが発生しないことが理論的に証明されている。

【0082】なお、本発明の乗算剰余計算装置は、ハードウェアに限らず、同様の機能構成の少なくとも一部をソフトウェアで実現することもでき、そのような場合にも処理の高速化を達成することができる。例えば、ソフトウェアで第2発明の乗算剰余計算装置を実現し、パイプライン処理が可能な32ビットプロセッサ上で実行した場合、乗算剰余処理時間を比較すれば、従来の方式の約半分の時間で処理が可能となり、その効果が明らかである。

【0083】**【発明の効果】**以上説明したように、第1、第2発明によれば、モンゴメリ法を用いた乗算剰余計算について、全ての処理を処理単位内で行えるので、DSP、マイクロコントローラ等の上のソフトウェアとして実現が容易になる。

【0084】また、第2発明によれば、キャリー伝搬を各積和毎に制限でき、一方の積和处理中にもう一方の積和处理も可能となるので、パイプライン処理などによって、高速に乗算剰余計算を行うことが可能である。

【図面の簡単な説明】

【図1】本発明の乗算剰余計算装置(第1発明)の構成図である。

【図2】積和回路の構成図である。

【図3】本発明の乗算剰余計算装置(第1発明)の動作説明図である。

【図4】本発明の乗算剰余計算装置(第2発明)の構成図である。

【図5】本発明の乗算剰余計算装置(第2発明)の動作説明図である。

【図6】モンゴメリ法による乗算剰余処理の一例を示す

フローチャートである。

【図7】コア前処理を行うための構成図である。

【図8】コア処理を行うための構成図である。

【図9】コア後処理を行うための構成図である。

【図10】積和回路の構成図である。

【図11】従来の乗算剰余計算装置の構成図である。

【図12】従来の積和回路の構成図である。

【図13】従来の乗算剰余計算装置の動作説明図である。

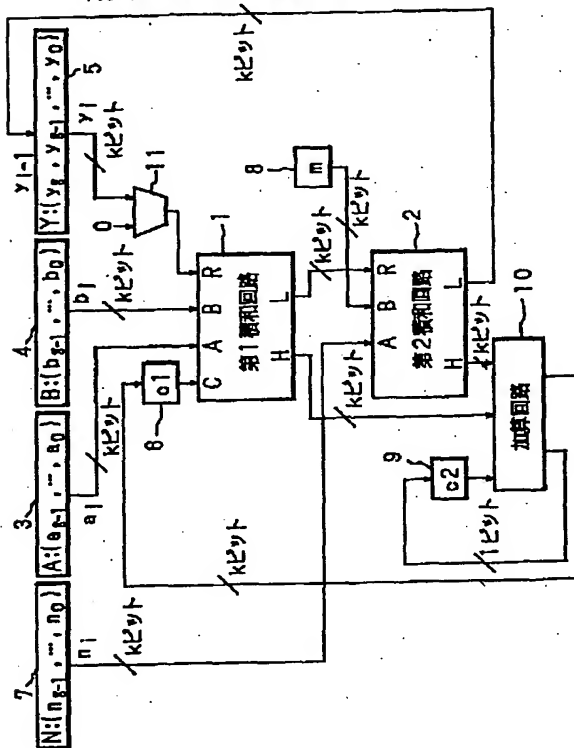
【符号の説明】

- 1 第1積和回路
- 2 第2積和回路

- 3 Aレジスタ
- 4 Bレジスタ
- 5 Yレジスタ
- 6 c1レジスタ
- 05 7 Nレジスタ
- 8 mレジスタ
- 9 c2レジスタ
- 10 加算回路
- 21 第3積和回路
- 10 22 第4積和回路
- 26 c3レジスタ
- 29 c4レジスタ

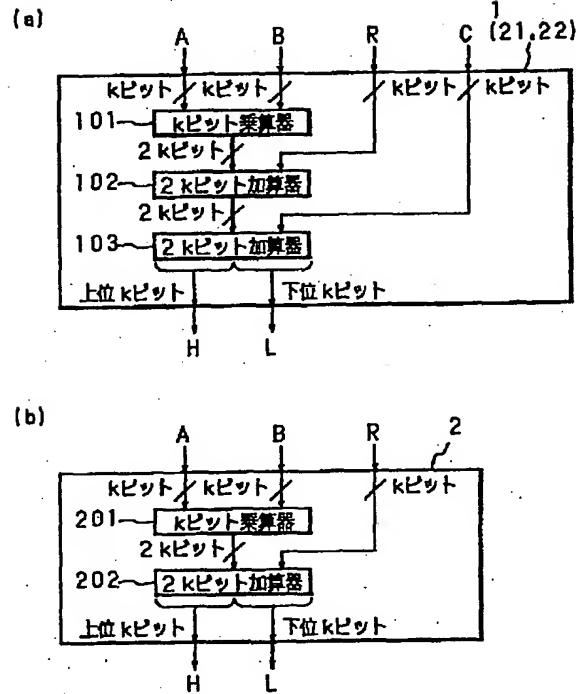
【図1】

本発明の乗算剰余計算装置（第1発明）の構成図



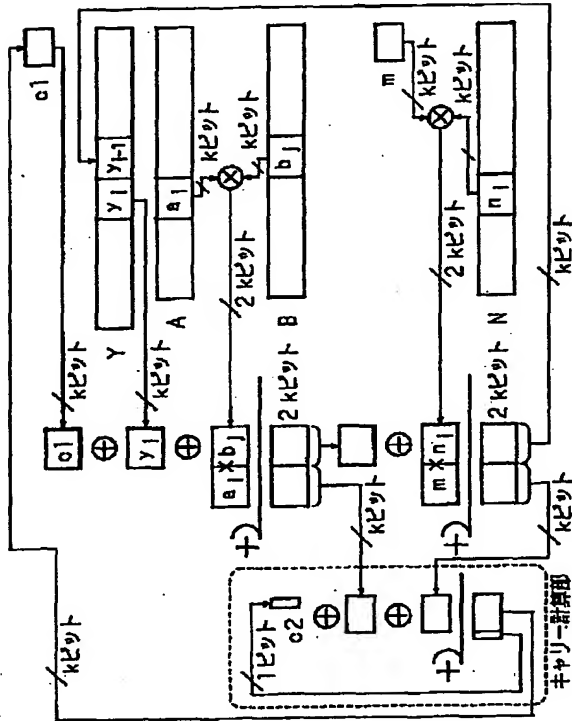
【図2】

積和回路の構成図



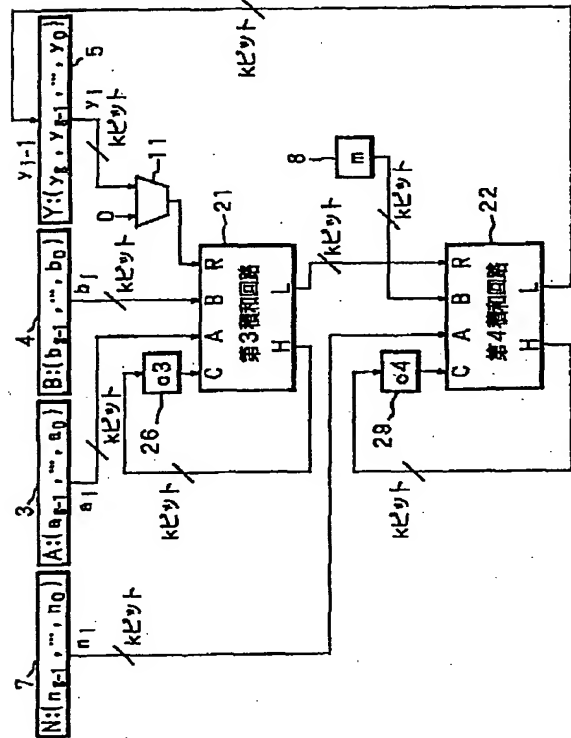
【図3】

本発明の乗算剰余計算装置（第1発明）の動作説明図



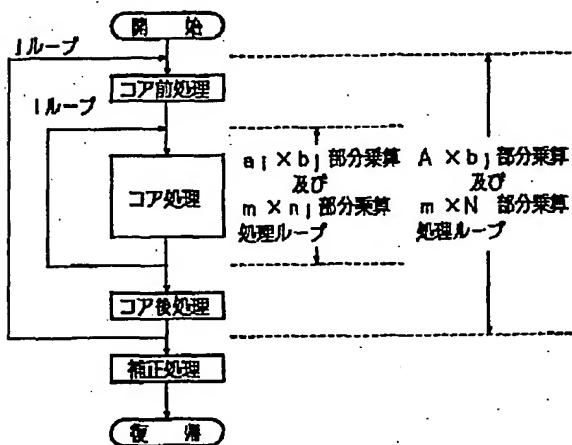
【図4】

本発明の乗算剰余計算装置（第2発明）の構成図



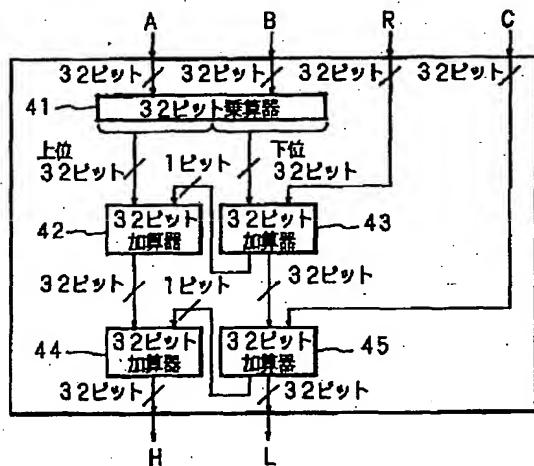
【図6】

モンゴメリ法による乗算剰余処理の一例を示すフローチャート



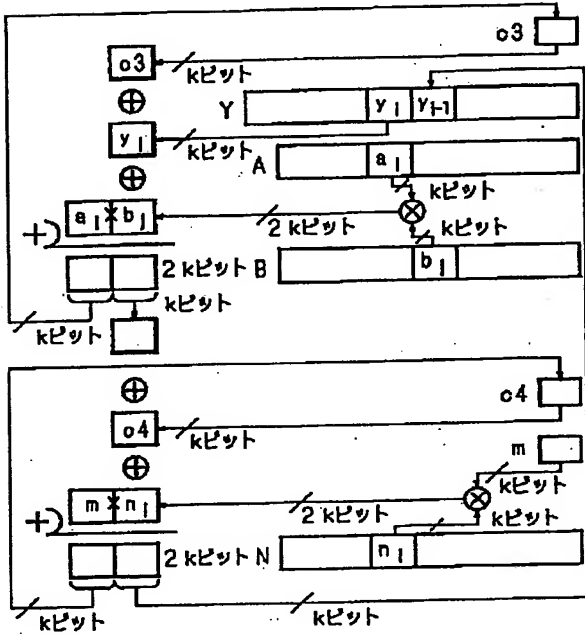
【図10】

積和回路の構成図



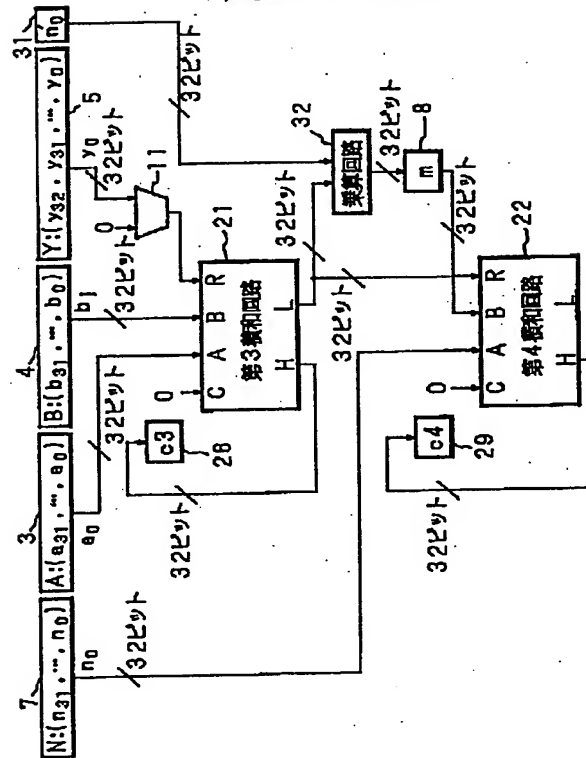
【図 5】

本発明の乗算剰余計算装置（第2発明）の動作説明図



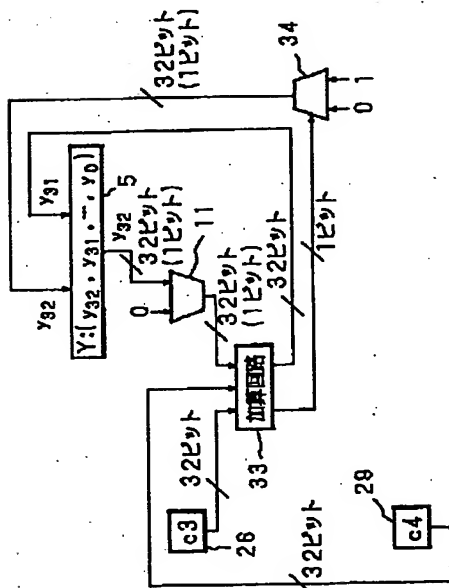
【図 7】

コア前処理を行うための構成図



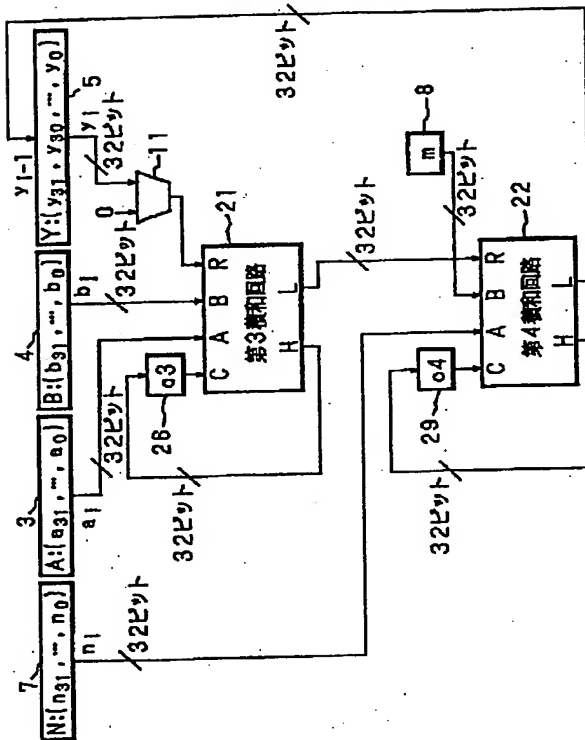
【図 9】

コア後処理を行うための構成図



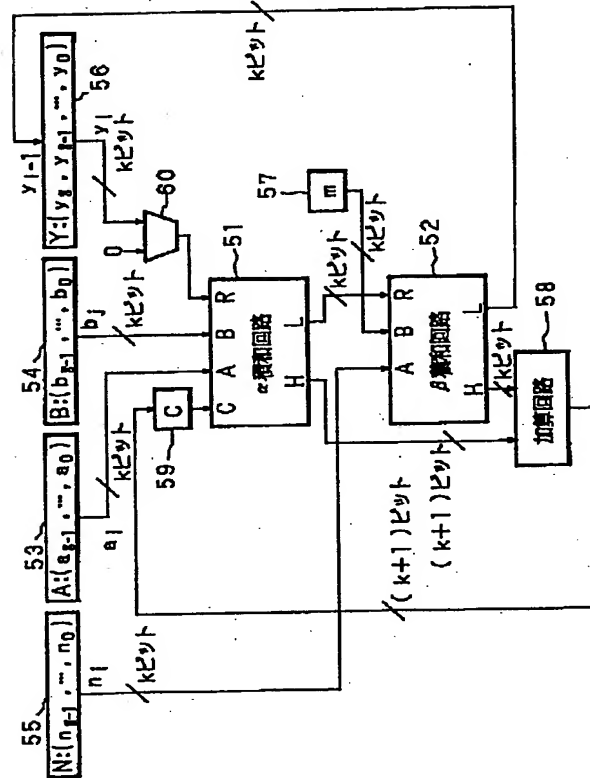
【图 8】

コア処理を行うための構成図



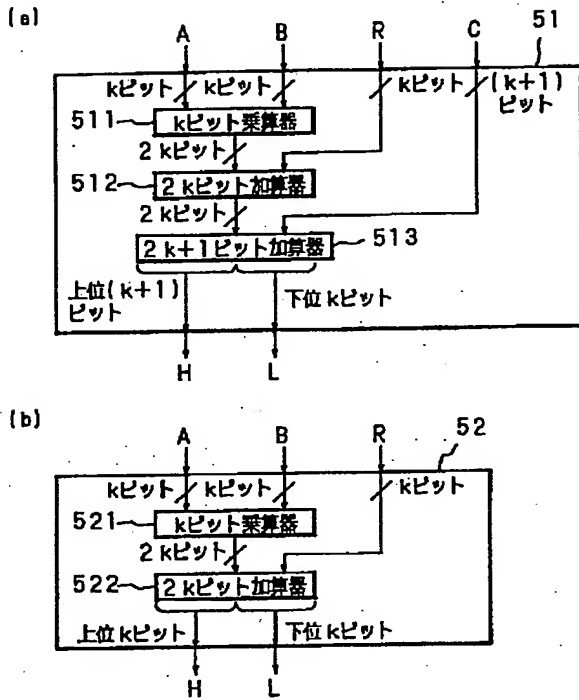
【図 1 1】

従来の乗算剰余計算装置の構成図



【図 12】

従来の積和回路の構成図



【図 13】

従来の乗算剰余計算装置の動作説明図

